1981

The Purdue Multimachine Pipeline: A High Bandwidth Machine Network and Programming Environment for Research in Large Scale Computation

Douglas E. Comer
Purdue University, comer@cs.purdue.edu

Peter J. Denning

John R. Rice
Purdue University, jrr@cs.purdue.edu

Lawrence Snyder

Report Number:
81-378

THE PURDUE MULTIMACHINE PIPELINE

A High Bandwidth Machine Network
and Programming Environment
for Research in Large Scale Computation

BY

Douglas E. Comer
Peter J. Denning
John R. Rice
Lawrence Snyder

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD-TR-378
September 1981

Abstract: We propose to implement and study a computer architecture that will provide an advanced programming environment and allow parallel execution of program parts on different machines.

In its early stages, the facility will provide a Multiprocess Viewport for managing data streams flowing between program parts; these streams will be observed, controlled, and edited through windows at the programmer's workstation. In the later stages, the Multiprocess Viewport will evolve into the Quanta programming environment, which permits hierarchical construction of software by plugging together program parts taken from an inventory.

Our final goal is the Multimachine Pipeline, which increases parallelism by executing logically connected program parts on different machines. The machines will be connected by multiple nets for the high bandwidth required to transmit large volumes of data between parts. The Quanta environment will provide the means of specifying programs for execution on the Multimachine Pipeline. We envision this facility as a prototype for future distributed systems, which should be extendable to large computational power in steps of one minicomputer.

The proposed facility will aid other research projects at Purdue in two ways. In the short term, the additional machines will provide a badly needed boost to computational power for existing projects; the prototype of the Multiprocess Viewport will assist all programmers who work with pipelined computations. In the long term, the project will provide a parts-based programming environment and will support large scale computation based on a large number of minicomputers rather than a single supercomputer. We expect that the resulting architecture can be replicated to provide similar environments elsewhere.

THE PURDUE MULTIMACHINE PIPELINE:

A HIGH BANDWIDTH MACHINE NETWORK AND PROGRAMMING ENVIRONMENT FOR RESEARCH IN LARGE SCALE COMPUTATION

Principal Investigators

Douglas E. Comer
Peter J. Denning
John R. Rice
Lawrence Snyder

Other Participating Faculty

Dennis Arnon
Janice Cuny
Dorothy Denning
Buster Dunsmore
Dennis Gannon
Christoph Hoffman
Tim Korb
Douglas McCarthy
Robert Lynch
Thomas Murtagh
Michael O'Donnell
Herbert Schwetman
Vincent Shen
Walter Tichy

September 10, 1981

Table of Contents

1. EXECUTIVE SUMMARY

1.1 Abstract

We propose to implement and study a computer architecture that will provide an advanced programming environment and allow parallel execution of program parts on different machines.

In its early stages, the facility will provide a Multiprocess Viewport for managing data streams flowing between program parts; these streams will be observed, controlled, and edited through windows at the programmer's workstation. In the later stages, the Multiprocess Viewport will evolve into the Quanta programming environment, which permits hierarchical construction of software by plugging together program parts taken from an inventory.

Our final goal is the Multimachine Pipeline, which increases parallelism by executing logically connected program parts on different machines. The machines will be connected by multiple nets for the high bandwidth required to transmit large volumes of data between parts. The Quanta environment will provide the means of specifying programs for execution on the Multimachine Pipeline. We envision this facility as a prototype for future distributed systems, which should be extendable to large computational power in steps of one minicomputer.

The proposed facility will aid other research projects at Purdue in two ways. In the short term, the additional machines will provide a badly needed boost to computational power for existing projects; the prototype of the Multiprocess Viewport will assist all programmers who work with pipelined computations. In the long term, the project will provide a parts-based programming environment and will support large scale computation based on a large number of minicomputers rather than a single supercomputer. We expect that the resulting architecture can be replicated to provide similar

environments elsewhere.

1.2 Development and Management of Facility

The principal investigators, Professors Comer, Denning, Rice, and Snyder, will oversee the development of the facility. Denning and Rice will act as co-directors during the first year while a permanent director is sought. Denning and Comer will select and acquire equipment. Comer will be primarily responsible for the Multiprocess Viewport project. Snyder and Rice will be primarily responsible for the Multimachine Pipeline project. A list of tasks appears in Section 5.6.

1.3 Research Projects

While the Multiprocess Viewport and Multimachine Pipeline are being developed, existing projects will benefit from the prototypes and increased computer resources. Section 6.1 describes the four principal beneficiaries:

Blue CHiP Architecture (L. Snyder)

Evaluation of PDE Software (J. Rice)

Equations Interpreter (C. Hoffman & M. O'Donnell)

High Order Methods for Elliptic Problems (J. Rice, R. Lynch)

Section 6.2 describes six projects that will support and enhance our approach to distributed computing:

Data Security (D. Denning)

Communications among Software Parts (W. Tichy & T. Murtagh)

Synchronization of Parallel Computation (J. Cuny & L. Snyder)

Performance Bounds of Multiprocessor Systems (H. Schwetman)

Programming Support for Personal Computer Networks (W. Tichy)

Interactive Programming Environments (T. Korb)

Section 6.3 describes four projects to validate our hypothesis that parts-based programming will significantly reduce programming time and errors:

Verifying Utility of Parts-Based Programming (B. Dunsmore)

Programming Effort for Very High Level Languages (J. Rice & V. Shen)

A Problem Solving Environment for Queueing Networks (H. Schwetman)

A Problem Solving Environment for PDEs (J. Rice)

Section 6.4 describes three projects that will investigate large scale scientific computation on the Multimachine Pipeline:

Symbolic Mathematical Computation (D. Arnon)

Networked Numerical Algorithms (D. Gannon)

Parallelism in Computational Fluid Dynamics (D. McCarthy)

A discussion of the budget and program management appears in Section 5.7.

EQUIPMENT SUMMARY

Description	unit cost	total cost	annual maint.
Year 1 –			
2 VAX 780 systems each with two 300Mb disks, 4Mb memory, 16 lines	250	500	50
2 Pronet network interfaces	4	8	--
YEAR 1 TOTAL		508	50
Year 2 –			
4 VAX 750 systems, each with 2Mb memory, 134Mb Winchester disk, BBN Bitmapped terminal	80	320	32
YEAR 2 TOTAL		320	32
Year 3 –			
10 Workstations (to be selected)	30	300	30
2 Ethernets with 6 interfaces	20	40	4
2 VAX 780 systems for pipeline	150	300	30
YEAR 3 TOTAL		640	64
Year 4 –			
10 Workstations	30	300	30
2 Ethernets with 6 interfaces	20	40	4
2 VAX 780 systems	150	300	30
YEAR 4 TOTAL		640	64
Year 5 –			
10 Workstations	30	300	30
2 Ethernets with 6 interfaces	20	40	4
2 VAX 780 systems	150	300	30
YEAR 5 TOTAL		640	64

(All Amounts in $K.)

2. OVERVIEW

We propose to implement and study a computer architecture that will provide an advanced programming environment and allow parallel execution of program parts on different machines. The project takes advantage of our faculty's strength and experience in programming environments, operating systems, and large-scale parallel computation. The resulting facility will further support these strengths in the years ahead.

The facility will be developed in three phases. The goal of Phase I (Year 1) is to upgrade the existing research facility of the department by adding two VAX 780 machines and a 10Mb local network to integrate them with the existing complex of 2 VAX 780. This will meet our severe, pressing need to significantly upgrade computing power. Four major research projects are falling behind schedule owing to insufficient computing power.†

The goal of Phase II (Years 2-3) is to complete work on the *Multiprocess Viewport* (MVP) for managing data streams flowing between program parts; these streams will be observed, controlled, and edited through windows at a user's display. Windows are a necessary tool for observing different portions of a pipelined computation. Stream control and editing are extensions that will provide new tools for testing and debugging stream-oriented computations. Each window will be connected to a *tap* on the data path between two parts selected by the user. A tap will keep a history file of the data flow; the user can edit and replay histories. By the end of Phase II, 10-15 workstations containing the MVP will be operational. The MVP meets a medium-term need for

---

†The projects are: 1) Blue CHiP, to study massively parallel computation in VLSI systems with reconfigurable switches between processing elements; 2) ELLPACK and TOOLPACK, to develop software parts for numerical computation and collect performance data on them; 3) Automatic Interpreter Generation, to generate interpreters for applicative languages from their algebraic specifications; and 4) Higher Order Methods for Elliptic Problems, to study the performance and accuracy of the HODIE method of solving Partial Differential Equations.

significant improvement in the quality of the programming environment.

The goal of Phase III (Years 4-5) is to complete work on the *Multimachine Pipeline* (MMP) and its programming environment, Quanta. We use the term pipeline in the sense "network of pipelines", not the restricted sense "linear pipeline". The Multimachine Pipeline will achieve parallelism not only by pipelining, but by parallel execution of independent programs.

The Multimachine Pipeline is a hardware environment and operating system for executing networks of programs connected by pipes. Parts can execute in parallel on different machines of different types. A high bandwidth network constituted of parallel Ethernets or fiber optics will be used to transfer data among parts. The Multimachine Pipeline will be feasible because the Quanta environment will provide the means to specify and initiate computations comprising a moderate number (say, 10-1000) parts. We believe the Multimachine Pipeline is a promising new approach to distributed computation, which must support general purpose program development and large scale computation, and which must be expandable in affordable steps to very large size.

The Quanta environment supports parts-based programming, a style in which one either applies software parts to data or connects software parts to form larger parts. [1] The Multiprocess Viewport provides the means to monitor and control executing parts. In Phase III we will complete work on the Quanta Parts Catalog, which will reside in a separate machine on the network, and on the Quanta parts composition system, which will allow programmers to graphically construct two-dimensional parts networks. By the end of Phase III, the parts composition software will reside in the workstations

[1] D. Comer, J. Rice, L. Snyder, and H. Schwetman, "Project Quanta," Technical Report CSD-TR-366, Computer Sciences Department, Purdue University (March 1981).

along with the Multiprocess Viewport.

The next section summarizes computer science at Purdue, showing the nature of existing facilities and demonstrating that our faculty is prepared to undertake the project. The section following that describes how the facility will meet our research needs. The section next after that describes the facility and its programming environment, including a time table of tasks required to reach the goal of each phase of the project. The final section describes the research projects that will be supported by the facility and how they will benefit from it.

3. COMPUTER SCIENCES AT PURDUE

The Computer Sciences Department at Purdue University is the oldest in the United States. It was founded in October, 1962, within the Mathematical Sciences Division of the School of Science. From the beginning, it has had strong research groups in numerical analysis, theory, and programming languages. In the early 1970s, the department extended its research to include performance evaluation and operating systems. By the mid 1970s research had grown to include software science and engineering.

3.1 Research Facility

Significant development in experimental computer science began in 1978 when we installed our departmental VAX running UNIX. We were the second site outside of Bell Labs to run UNIX 32/V. (Berkeley was the first.) This facility supported new work in text editors, programming environments, compilers, and interpreter generators. We have continually expanded this facility. By the end of 1981, it will be configured as shown in Figure 1. This facility has been financed with about $115K of research funds, $105K of industrial funds, and $520K of university funds.

We have close ties with the computer systems group in the Electrical Engineering Department and with the CAD Laboratory graphics group in the Mechanical Engineering Department.

3.2 Faculty

Our faculty presently stands at 27 members representing 22 Full Time Equivalents; 7 are full professors, 8 are associate professors, and 12 are assistant professors. Seven of the assistant professors specialize in systems and languages. They form an important component of our effort in experimental computer science, and they complement the strength of our senior faculty.

One measure of the proven research strength of the faculty is the level of grant support they have attracted. At present, 17 members of the faculty are principal investigators in 22 separate projects, totalling about $2.6 millions (about $1.3 millions annually). The projects are listed in Appendix A. We are also successful in attracting funds from industry; in the past year, the first in which we made serious attempts to

Equipment	Investment
2 VAX systems with total of 7 megabytes main memory, 1360 megabytes disk, and 50 terminals	$550K
Megatek interactive graphics system with PDP 11/20 front end	$60K
Tektronix 4112 high resolution graphics terminal and software	$30K
Pronet 10Mb local network	$15K
3 BBN Bitmapped display terminals	$15K
Peripherals: 2 Diablo printers, 1 Versatec printer, 1 HP 7221B multicolor plotter.	$65K
5 LSI-11 computers (lab)	$45K
Racal/Vadec telecommunications equipment	$7K
TOTAL:	$743K

FIGURE 1: Summary of Departmental Equipment

raise outside funds, our industrial friends have contributed about $100K.

Another measure of our strength is our graduate students. There are currently 140 graduate students, of whom 25 have passed the PhD qualifying examination. The department awarded 33 PhD degrees between May 1975 and August 1981. The department has been a member of the Bell Laboratories and Sandia Laboratories One Year On Campus (OYOC) masters programs since their inceptions. Most of the masters degree

holders accept jobs in industry: significant numbers of Purdue graduates hold jobs at Bell Laboratories, TRW, Intel, Hewlett Packard, IBM, Univac, Control Data, and other major computer manufacturers. We are particularly proud that four of our recent PhDs hold key positions in the Intel iAPX 432 computer architecture project.

3.3 Accomplishments

The principal investigators have accumulated considerable experience that prepares them to undertake the construction of a major research computing facility.

Doug Comer was director of the department VAX UNIX facility from 1979 through 1981, and is now a principal investigator on the Purdue contract of CSNET. He has deep knowledge of UNIX, local networks, programming environments, compilers, and interactive graphics. He deals with both hardware and software. Before directing the VAX facility, he had written and successfully distributed several major software systems. As director he oversaw several major UNIX development projects, including rewriting of most of the tty, disk, and other device drivers for UNIX; modifications of local UNIX software; and the first working Pascal compiler for UNIX in the U.S. He has specified and selected most of the major components of the facility.

Peter Denning has been head of the department since 1979 and is also a principal investigator on the CSNET project. He has long experience with high-level architectures of operating systems and with performance evaluation. He is an internationally recognized leader in the field, having received many honors for his technical contributions.

John Rice has long experience with mathematical software, dating back to the NAPSS (Numerical Analysis Problem Solving System) in 1965. He is the "founder" of the mathematical software discipline and has been heavily involved in software

performance evaluation, parallel computation, selection of software for problem solving, the ACM Algorithms service, and traditional numerical computation. He is currently directing ELLPACK, a large project to develop and evaluate software for elliptic partial differential equations. He is also a member of the TOOLPACK group for portable Fortran software tools.

Larry Snyder, the director of the Blue CHiP project, is studying reconfigurable, highly parallel processor arrays that are implementable on VLSI chips. His broad background includes the theory of parallel computations, data structures, data base systems, and APL computing. His experience in constructing large software systems, simulators, and language processors is extensive; in the past year, for example, he has directed the construction of two production compilers (LAPSE and Config) and a parallel architecture simulator. He is also director of the department's VLSI design laboratory.

The principal investigators will be assisted by the following personnel, who will play key roles in the development of the facility:

- Bob Brown is an expert in the UNIX system and with hardware and communications systems. He has built a prototype window manager which will shortly be operational on the Tektronix 4112 terminal and later on the BBN bitmapped displays.
- Dennis Gannon is an expert in large scale computation and the design of VLSI hardware. He was one of the four new researchers to be funded in 1981 by the NSF New Faculty Investigator Program.
- Tim Korb has extensive background as a software engineer, including his graduate study at the University of Arizona and two years at Xerox PARC, where he worked on the Bravo-X editor and other interactive systems based on bitmapped displays.

- Walter Tichy's experience as a software engineer includes a role in the Carnegie-Mellon University Gandalf project. He is currently working on configuration and version management database systems for large software projects.
- Herbert Schwetman has extensive experience with the Purdue University Computing Center as a systems programmer and performance evaluator. He is currently the director of the department VAX facility.

In addition, our group works closely with the Electrical Engineering computer systems group (mainly with George Goble and Mike Marsh). Their notable recent accomplishment is the dual-CPU VAX, which permits a large increase in the computing power of the machine at a third of the cost of a new mainframe. We also work closely with the Mechanical Engineering CAD Laboratory (mainly with David Anderson, who holds a joint appointment in this department) on our graphics display systems.

3.4 Summary

We have a strong, young faculty, many gifted students, a basic research facility, and the backing of our administration. Many research projects are already under way. Many of the faculty have wide recognition in their fields. With a new infusion of resources, we have the potential to develop an outstanding experimental computer science research effort and to contribute new approaches to large-scale computation.

4. RESEARCH NEEDS

4.1 The National Situation

The problems faced by computer science departments across the nation have been analyzed in the Feldman Report[2] and the Snowbird Report[3]

- Industrial salaries and facilities have become so attractive that a high percentage of bachelor's recipients opt for immediate employment instead of graduate school (70% in 1980 compared with 30% in 1975); that diminishing numbers of qualified masters recipients opt to continue toward the PhD; that PhD recipients opt for industry rather than academia; and that faculty switch to industry in much greater numbers than industry employees switch to academia.
- Working conditions for the remaining faculty are becoming intolerable because of steadily increasing numbers of majors with virtually no growth in faculty size or modernization of facilities. The faculty have insufficient time for proper supervision of graduate students or for research.

This problem threatens the national well-being in three ways:

- Without larger numbers of qualified faculty, we cannot meet the demand for computer specialists for the rest of the 1980s.
- Without better facilities and additional time for supervision, faculty cannot properly teach their students. Universities will be unable to fulfill the principle that the teachers should also be practitioners of the art (i.e., experimenters) in a discipline where experimentation is essential to learning.

---

[2]J. Feldman, Editor, "Rejuvenating Experimental Computer Science," *Communications of ACM*, September 1978.

[3]P. Denning, Editor, "A Discipline in Crisis," *Communications of ACM*, June 1981.

- Universities perform almost all the public-domain experimental computer science research there is in the United States. If this function ceases, the free flow of frontier information through journals and public symposia will be stanched. This poses a serious threat to the nation's ability to maintain its world lead in computing technology.

4.2 The Purdue Situation

The problems faced by computer science at Purdue are instances of the national ones. The administration of Purdue University is committed to a long-range plan to maintain our excellence and contribute to the solution of these problems. An important component of this plan is the research facility. The current facility must be significantly improved in both power and quality to support our active groups working in programming environments, program measurement, large-scale numerical computation, system performance, and parallel computation. The proposed facility will accomplish this by providing advanced computing equipment, ready assistance in developing hardware and software, and professional facilities management.

The proposed facility will meet our needs for increased power and a better programming environment in three stages:

1. **Short Term (1 year):** The current VAX system supports about 50 researchers. It is overloaded. Our major research projects are beset by a severe shortage of computational power. Because the University Computing Center does not offer UNIX-based research computing, we cannot support most of our research from non-departmental sources. Our goal in Phase I of the project is to upgrade departmen-

tal facilities by adding two VAXes to the department's network.

II. **Middle Term (2-3 years):** The quality of computing will be improved by improving the user interface. Our goal in Phase II is to create the Multiprocess Viewport (MVP), which will support the construction and debugging of pipelines composed of software parts. The MVP will provide the ability to observe, edit, and restart data streams flowing between program parts. It will eventually reside in the user's workstation.

III. **Long Term (4-5 years):** Our goal in Phase III is to create an environment that will achieve our final objective of supporting parts-based programming and large-scale distributed computing through an expandable network of machines. A graphical parts-composition system will extend the MVP and an inventory of certified software parts will greatly reduce the effort of programming; this will constitute the Quanta programming environment. Through Quanta, programmers will be able to specify computations consisting of software parts that will be executed in series and parallel on a network of minicomputers (VAXes or equivalent); this network and its operating system will constitute the Multimachine Pipeline. It will also provide access to other networks, databases of software, and sophisticated output devices.

This facility is intended to support the types of research at which we already have considerable expertise and expect to continue. The Blue CHiP VLSI project requires the MVP to construct and debug the software used to design large, reconfigurable arrays of processing elements; the ability to observe data streams will enhance the designer's ability to debug stream-oriented algorithms before committing them to chips. The ELLPACK project is deeply involved in modular software for solving partial differential equations; the MVP, and later the Quanta parts-composition system, will signficantly aid this effort. Our programming environments and software metrics

projects will quantify the benefits achievable with a very-high-level environment based on program parts. Our programming language projects will formalize new principles of program construction, type checking, and data security for an environment in which data streams are the primary objects of computations. These and other projects that will benefit from, and support, the facility are discussed in greater detail in Section 5.

The proposed facility will help the department build a highly attractive research environment. It will support the kinds of projects at which we are best and have proven reputations. It will also help other institutions, as well as other groups at Purdue, by providing a new approach to large-scale computation.

5. PROPOSED RESEARCH FACILITY

The proposed facility is organized around the two broad areas of research among our faculty -- program construction principles and large scale computation. The facility will start simple and evolve within five years to a distributed system coordinating a complex of computers and a network of personal workstations, all within the programming environment we call Quanta. All hardware acquired will be used immediately to support research; the Quanta environment will be evolved gradually. The three phases of the development have these goals:

Phase I (Year 1)	--	Acquire and install two new VAX 11/780 as part of the existing departmental facility. (Select prototype workstations and bring up experimental prototypes of the Multiprocess Viewport for Phase II. Begin planning the Multimachine Pipeline for Phase III.)
Phase II (Years 2-3)	--	Bring the Multiprocess Viewport into full operation on workstations. (Bring up the prototype of the Multimachine Pipeline and its operating system for Phase III.)
Phase III (Years 4-5)	--	Bring the Multimachine Pipeline into full operation. Complete work on the programming environment.

The next section contains an overview of the programming environment envisioned for the Multimachine Pipeline. Next are three sections giving details of the components developed in Phases I, II, and III. Last is the time table of tasks required to complete the project.

5.1 A Programming Environment for the Multimachine Pipeline

Parts-based programming is an approach that has great potential for increasing the productivity of programmers. It is a sound framework for expressing distributed computation. In this style of programming, one either applies software parts to data or connects software parts together to form larger parts. A software part is any program whose computation, precision, performance, and environmental requirements are completely and unambiguously described in the specifications. Each part certifies its input, rejecting any data that lies outside the specified domain.

Our approach to this environment is called Quanta. It draws heavily on familiar intuitions from electronics parts technology. Program parts will be described by catalog pages. They will be retrieved from inventory and plugged together using an

interactive graphics system. They will be used by connecting them to input and output sources and activating them.

These three intuitions correspond to the three major components of the Quanta environment. The first component is a database called the *parts catalog*. Each record in this database contains keywords and phrases, a catalog page, and a pointer to the part. A catalog page includes a black-box picture showing inputs and outputs; it specifies the types of data that may flow in and out; it describes the resource requirements; it states any other assumptions about the environment that must hold for the part to operate correctly; and it describes the kind of certification and validation (for quality control) the part has undergone before being placed in the inventory. A query system will assist the user locate parts by function.

The second component is a *parts-composition system*. It is the environment in which a user will construct networks of parts, called *compositions*. He will select parts from the catalog and compositions from the file store, position pictures of them on his interactive graphics screen, and draw connections between them. He can run the composition on test data and use the results to optimize the connections and locate errors. He can name the composition and save it for later use. He can submit a composition for certification and subsequent addition to the parts catalog.

A simple, familiar form of composition is the linear pipeline of UNIX. We write

$$H \text{ file} \mid G \mid F$$

to denote the functional composition

$$F(G(H(\text{file}))) .$$

In Quanta, we would represent this by a directed graph:

$$\text{file} \rightarrow H \rightarrow G \rightarrow F \ .$$

Quanta will interpret the graph as a data flow graph rather than a functional composition. This means that the various elements can all be operating in parallel as data streams through the graph. General program compositions in Quanta will be called *Quanta graphs*. Initially they will be directed acyclic graphs, but general graphs are possible later. Programmers will draw Quanta graphs on a display; experience with two-dimensional pipelines at Bell Laboratories[4] and University of Arizona[5] suggests that powerful functions are possible with pipeline networks but that linear notations are restrictive.

Figure 2 is the Quanta graph for a problem of the type studied in Rice's ELLPACK project -- collecting data on the performance of subroutines. The computation is organized as a pipeline whose parts can process data streams; each part is assigned to a separate machine. Note that different machines can be used, according to the needs of the parts.

The third component of Quanta is a set of *problem solving environments* (PSEs). A PSE permits a worker in a discipline to use software for solving problems in that discipline without having to be a programmer. A PSE knows the notation and terminology of the discipline. An example in current technology is an APL system. An example in Quanta technology is a performance evaluation PSE: its user can draw queueing network diagrams of computer systems, calculate the values of parameters from performance data, validate models, and make predictions about the performance of future networks. Output generated by a queueing network model can be displayed in graphs

---

[4]D. McIlroy's group; no report published.

[5]Fraser, C. W., "A software system and command language based on connecting coroutines," Technical Report 80-17a, Department of Computer Science, The University of Arizona (July 1980).

and tables familiar to performance analysts. The user need not know how the algorithms work; they will be parts invoked by the PSE on his behalf. We believe that the concept of PSE is essential to allow wider use of quality software without requiring everyone to become a trained programmer. [6]

Quanta combines proven concepts and new ideas. Embrionic parts catalogs already exist: the ELLPACK and IMSL libraries of mathematical software and many programs listed in the *UNIX Programmer's Guide* embody these ideas. The concept of plugging parts together is well tested for linear connections in the form of UNIX pipelines and shellscripts.

We believe that parts-based programming will raise the level of language for expressing distributed computation and will permit nonprogrammers to use software reliably. Some of the research to be supported by our facility will test this hypothesis.

5.2 Phase I – Immediate Equipment Upgrade

Figure 3 shows the expected configuration of the departmental research facility by the end of 1981. The present facility consists of two VAX 11/780 (A and B in the figure), various peripherals, links to the Purdue University Computing Center (PUCC) and the Electrical Engineering (EE) networks, and a Megatek interactive graphics system. The VAXes and Megatek machines will be connected on a Pronet 10Mbaud Star Ring network.

We propose to add two additional VAX 11/780 systems (C and D in the figure) on the local network. They will be used as slave machines, running large jobs downloaded

---

[6] Rice, J. R., "Programming Languages: Power, Trends and Facilities for Numerical Computation," in *Relationship Between Programming Languages and Numerical Computation* (J. Reid, ed.), North-Holland (1982).

from VAX A or VAX B.

5.3 Phase II -- Multiprocess Viewport

The Multiprocess Viewport (MVP) is a generalization of the virtual terminal (VT) concept. It will be coordinated with the UNIX pipeline mechanism to permit observation, control, and modification of data streams flowing between software parts. It will aid in incremental program construction and in debugging.

The usual virtual terminal facility is a method of multiplexing several input/output streams to the display and keyboard of a single terminal. [7] The portion of the display allocated to one of the virtual terminals is called a window; it acts as the standard input and output for a running program. (See Figure 4.) The window containing the cursor is called the current window. Input from the keyboard is directed to the program connected to the current window (and is echoed in that window). Output from a program is immediately displayed in its window. The interface seen by a program is the standard UNIX interface.

The MVP interprets each program as a process acting on a data stream. It uses the virtual terminal concept to observe output and supply input through windows of the display. But it adds new, *stream control functions* that allow backing up, changing stream elements, stepping forward, and "thumbing" (selecting stream positions by specifying numbers between 0 and 1). With the MVP, the user can activate a *tap* on any pipe connecting two processes. Each tap sends a copy of the data stream to a viewing process, whose output is connected to a window on the user's display; each viewing process formats the display according to the expected data type at the tap. Data will be

[7] Teitelman, W., "A display-oriented programmer's assistant," Technical Report CSL 77-3, Xerox Palo Alto Research Center (March 1977).

sent between an MVP and its associated set of taps via a local network.

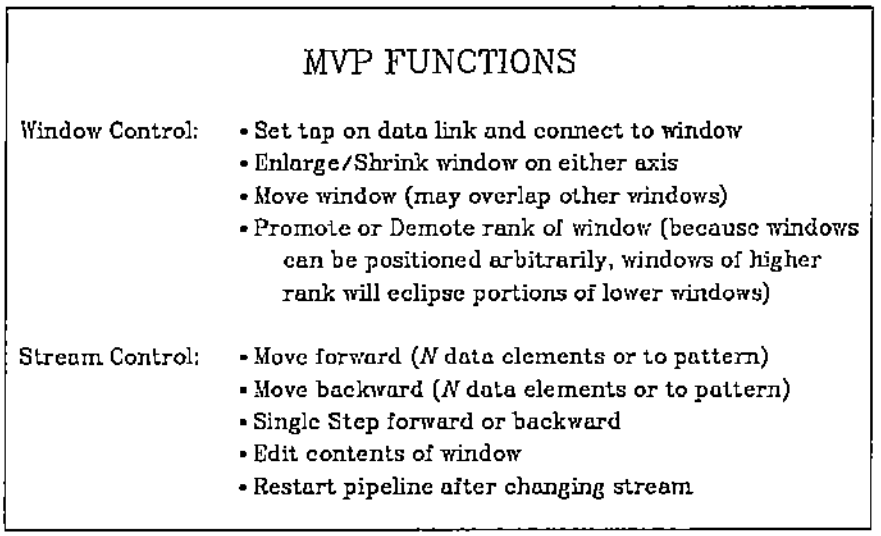

FIGURE 5: Functions of Multiprocess Viewport.

Figure 5 lists functions to be provided by the MVP. Figure 6 illustrates the connection between the taps and the MVP facility. Figure 7 illustrates incremental construction of a pipeline.

A prototype of MVP that handles only the window management functions will be started in Phase I. We expect to exploit some new data structures developed by Professors Comer and O'Donnell to improve the performance of window generation on bit-mapped displays. During Phase II we will redesign pipes so they can be tapped; we will implement and the stream control functions of the MVP in the host machine. At the end of Phase II we will move these functions into a workstation. (The workstation will also contain an editor and a cache of functions used frequently by the programmer.)

The *tapable pipe* will replace the ordinary pipe the data path between two other programs. When activated, the tap sends a copy of the data stream to an attached viewing process in the MVP. It keeps a copy of the stream in a *history file*, which can

Example: To preview formatted output of a text file at his terminal, the user sets up a pipeline

tbl file | eqn | nroff

On being satisfied that file is in correct format, he changes the last element to 'troff':

tlb file | eqn | troff

Because the stream from 'tbl file | eqn' is saved, it can be fed into the new pipe terminator, 'troff', without reexecuting prior portions of the computation.

Example: Incremental construction of pipelines can proceed as follows. Suppose

$$F_1 \mid F_2 \mid \cdots \mid F_n$$

works for a given data stream. The user adds the next stage,

$$F_1 \mid F_2 \mid \cdots \mid F_n \mid F_{n+1}$$

observes the output, disconnects F_{n+1} if necessary to repair, reconnects it, passes the saved stream through F_{n+1} again, and so forth, until F_{n+1} is satisfactory. There is no need to recompute the result of the prefix $F_1 \mid F_2 \mid \cdots \mid F_n$.

FIGURE 7: Examples of incremental parts-based programming.

be changed and replayed on command from the viewing process. The tapable pipe part is a generalization of the ordinary UNIX pipe feeding into the *tee* program; *tee* makes a copy of the file that passed beween two other programs, but does not allow modification or replay of the computation.

There are two approaches to restarting a pipeline after a change in the history file at a tap. The first (Version 1) is simply to reinitialize all parts downstream and replay the history file from the beginning. This has the potential disadvantage of repeating substantial amounts of computation. The second approach (Version 2) uses taps that

periodically set checkpoints by recording in their history files the internal state of their immediate downstream neighbors. To restart the pipeline, the tap puts its immediate downstream neighbor into the nearest recorded state before the change in the history, signals the other downstream taps to do likewise, and replays the stream from that point. We will experiment with both kinds of tap, but will use only Version 1 in our early designs.

We intend that the MVP workstation be an alternative to the UNIX shell. Users can continue to use UNIX from ordinary terminals, e.g., on remote dialin. If its tap is not activated, the new pipe will behave exactly as the old.

Prototypes of these facilities already exist at Purdue. Doug Comer has implemented a UNIX program called *pg* (for "paginator"). Like other UNIX paginators (e.g., Berkeley's *more*), *pg* allows the user to step forward one or more screenfuls at a time and to search forward for a pattern. Unlike other paginators, Comer's *pg* allows the user to search *backward* for a pattern. The Version 1 tapable pipe will generalize *pg* by allowing the saved stream to be modified and retransmitted to subsequent programs.

Robert Brown has implemented a prototype of the window manager for the MVP, using a program similar to the UNIX *tee* program and Adds-40 terminals. The Adds-40 terminals are too slow to handle more than two windows at a sufficient speed. Shortly after the Tektronix 4112 terminal is delivered (September 1981), Brown expects to have his prototype running on it. We will also experiment with this facility on the BBN bit-mapped displays that will be delivered in Winter 1981-82.

The machine to be used for the workstation will be selected in Year 3 of the project. Our experience with the prototype MVP during Years 1 and 2 will be necessary to guide us. By 1983 there will be several candidate machines available. Figure 8 lists several of which we are aware. Each will have a local disk store and bit-mapped display; prices vary according to the memory size and resolution of the display.

CANDIDATE WORKSTATIONS

Machine	Estimated Cost
Xerox Dolphin	$60K
HP (68000-based)	$60-72K
Sun Terminal Workstation	$30K
Apollo	$30K
PERQ	$30K

VAX FAMILY MACHINES

Serial Number	Name
790	Venus
780	Starlet
750	Comet
730	Nebula

FIGURE 8: Candidates for workstations.

Figure 8 also lists a family of VAX machines that should be available in Year 3; the VAX 730 is also a candidate for a workstation.

For the prototype workstation, started in the first year of the project, we are attracted to a VAX 750 with Winchester Disk and BBN Bitmapped display (about $80K at current prices). This choice has several advantages. The VAX 750 can run UNIX, which would greatly reduce the development time of the MVP; moveover, the MVP may be movable to the VAX 730. Because the VAX architecture is compatible with all current software, all current programs would be able to run as well as MVP. When not in use as workstations, these VAX 750s will be available as additional machines on the network. Given the rapid advances in workstation technology, we wish to defer the final decision

until we can evaluate the machines that will be available.

To summarize: the main work in Phase II is to refine the design of the window and stream controllers based on experience with the prototype; to select and acquire machines and displays for workstations; to move the MVP from the host UNIX to the workstations; to design the tapable pipe; and to select and implement a communications protocol between the taps and viewing processes.

5.4 Phase III – The Multimachine Pipeline and its Programming Environment

It is time to lay the groundwork for an important class of large scale computer systems a thousand times more powerful than present "supercomputers" (such as the Cray-1 or IBM 3081). This class of systems will support interactive program development and will be upgradable in moderate steps to very large sizes. These computers will consist of many machines of different types connected to a high-bandwidth network. Though long desired, systems of this type have not been built because of two factors:

- **The data movement problem:** When a large computation is broken into a moderate number of parts, the amount of data to be moved between parts tends to be large. There has been no available technology for moving large volumes of data quickly among different machines.†
- **The programming problem:** There has been no convenient method of specifying a computation as a moderate number (say, 10-1000) of pieces, then coordinating the executions of the pieces on different machines of different types.

†Two approaches have been used to avoid the data movement problem. One is to express the computation as the interaction of a large number of small processes that need to exchange only small amounts of data; this approach is under active study by VLSI research groups. The other is to store all the pieces in a common memory and run them on different processors; memory contention limits the effective parallelism.

The Multimachine Pipeline (MMP), which exploits the high speed communications technology now coming to market, is a solution for the first problem. The Quanta programming environment is a solution to the second problem.

Figure 9 shows an architecture for the network connecting the machines. The desired bandwidth can be achieved by a multi-net, which is a set of parallel nets (the figure shows five nets, but any number could be used). The interface will be designed so that a machine initiating a transmission will select a net not already engaged by that machine.

Not every machine may be connected to every net (e.g., when there are more nets than ports on the multinet interface). One net should, however, be connected to every machine so that the machine initiating a computation can distribute the parts. It is necessary only that logically connected parts be allocated to physically connected machines.

Two remaining components of the Quanta environment will be assembled during Phase III. In Year 3 we will begin setting up a parts inventory. The INGRES data management system of Berkeley UNIX will be the medium and the UNIX, ELLPACK, and IMSL libraries will be reviewed for an initial stock of parts. In Year 4 we will complete work on the parts-composition system, which will allow the programmer to draw Quanta graphs on his display (using parts from the inventory), set taps in them for use with the MVP, and execute them. Initially, the parts-composition system will be constructed on the Megatek graphics system, but it will reside ultimately in the user workstations. An important problem whose solution will be due by this time is the *type-checking problem:* the system must warn the programmer if an incompatibility exists in a path he has specified from one part's output to another part's input. The solution must specify how types are represented and declared and how a part is

assured that the data presented is of the correct type.

Work on the final component of the Quanta environment, the Problem Solving Environments, will be conducted as side projects. ELLPACK is a working prototype for Partial Differential Equations. [8] This work will not be part of the facilities development.

When the user calls for the execution of a Quanta graph, the operating system will assign the components of the graph to different, appropriate machines. This assignment will take into account specifications from the programmer about the class of machine needed for each part. It will also take into account the physical data paths between machines. The tapable pipe mechanism will be modified to transmit the data stream over a network to the destination part in another machine.

The Multimachine Pipeline and its programming environment will mitigate two other common software problems. First, it will help construct programs from modules written in different languages: the uniform interface of the tapable pipe will allow such modules to communicate smoothly. Second, it will diminish the need to transport complex software to new machines: a part will be loaded only into machines for which it is certified.

To summarize: the main work in Phase III is to get the initial software parts inventory and parts composition system operational, to procure additional machines and Ethernets (or other high-bandwidth local nets), to redesign UNIX pipes so that the two ends can be in different machines, to implement algorithms that map Quanta graphs to the machines, and to implement a schedule and loader that distribute the parts of a Quanta graph to the assigned machines.

---

[8] Rice, J. R., ELLPACK: Progress and Plans, in *Elliptic Problem Solvers* (M. Schultz, ed.), Academic Press, (1981), 135-163.

5.5 Performance Evaluation of the Facility

Although most of the development of the facility will be the application of known principles, some of the work will require development and careful evaluation of new principles. We are therefore committed to measuring every component of the facility at every stage of development. The resulting data will be used to evaluate design alternatives, to match the equipment configuration to the load, and to measure the extent to which the facility achieves the projected benefits. Our data and experience will be reported to the community so that others may benefit from our work.

Some of the system behaviors that we will evaluate by measurement are: performance improvements achieved with new algorithms for bitmapped displays; optimal size cache of program parts in workstations; data traffic generated on the network by the Multiprocess Viewport; data traffic generated on the network by flows between program parts in the Multimachine Pipeline; contention between these two types of traffic; peformance of schedulers and loaders for Quanta programs on the Multimachine Pipeline; overheads of different methods of data type checking between parts; and volume of data transmitted between parts. The measurements and associated modeling work will be conducted by Professors Denning, Schwetman, and Tichy.

The Quanta programming environment embodies a hypothesis that parts-based programming will foster significant improvements in programming productivity. Our software metrics group, headed by Professors Conte, Dunsmore, and Shen, will conduct controlled experiments to evaluate this hypothesis. These experiments will compare Quanta with conventional programming environments with respect to total time and errors in completing projects. They will attempt to isolate the factors in the Quanta environment that are most strongly correlated with reductions in programming time or errors.

5.6 Time Table

The following is a presentation of the tasks for each phase of the project. Individuals reponsible for tasks in the first two phases are shown. Equipment acquisitions are indicated in the right column. Note that Phase 0 shows tasks that will be accomplished before the grant period begins.

PHASE 0 (1981-82):

a)	A prototype of the window control of the Multiprocess Viewport runs on the Tek 4112 and BBN Bitmapped displays. (Version 1 of MVP operational.) [Brown, Comer, Denning, Korb]	Tektronix 4112 display 3 BBN Bitmapped terminals
b)	Departmental machines are integrated on local network. [Schwetman]	Pronet local network

PHASE I (1982-83):

a)	Acquire and install two new VAX 780 and Ethernet. Connect research computers (4 VAX, peripherals) on Ethernet; integrate with other campus facilities and CSNET. Operate the VAX 780 in slave mode on the net. [Schwetman]	2 VAX 11/780 DEC Ethernet
b)	Acquire 4 VAX 750 for prototype workstations. Transport MVP window controller to this machine using the bitmapped display system. [Korb, Brown]	4 VAX 11/750 4 Winchester Disks 4 BBN Bitmapped displays Appropriate Ethernet equipment
c)	Design Version 1 tapable pipe (this version does not save state of downstream part in its history). Design communications protocol between the tap and its attached viewing process in MVP. [Comer, Denning, Korb, Snyder, Tichy]	

d) Design MVP stream control functions for Version 1 tapable pipe.
[Comer, Denning, Korb, Snyder, Tichy]

e) Begin exploratory work for the Multimachine Pipeline -- i) communications protocols for pipeline elements, ii) kernel modifications to UNIX so ends of pipe can be in different machines, iii) algorithms to map Quanta graphs to machines.
[Comer, Gannon, Snyder, Tichy]

PHASE II (1983-85):

a) Implement Version 1 tapable pipe and MVP stream control functions and test on host machine.
[Comer, Korb, Tichy]

b) Implement communication protocol between tapable pipe and MVP viewing processes via Ethernet.
[Comer, Korb]

c) Move MVP to VAX 750 workstation. (Version 2 of MVP now operational.)
[Korb]

d) Select and acquire machines to be used as workstations. Install on the local network. Install MVP and local editor in each workstation.
[Comer, Korb]

10 workstations (Year 3)
Ethernet interfaces

e) Design protocol for interfacing a machine to the Multi-Ethernet as part of the Multimachine Pipeline. Acquire two additional Ethernets and two VAXes for the initial version of the MMP. (Prototype MMP operational by end Year 3.)
[Gannon, Korb]

2 Ethernets
2 VAX for pipeline (Year 3)

f) Design algorithms for mapping Quanta graphs onto the machines of a pipeline, and design a loader that puts parts on these machines for execution. Complete design of part-part communications protocol is due by this time.
[Comer, Gannon, Snyder, Tichy]

g)	Design Version 2 tapable pipe and explore consequences on the design of parts. Install in MVP and test. [Tichy, Rice]	
h)	Begin establishing Quanta parts inventory. Review UNIX, ELLPACK, and IMSL libraries to identify programs which are already parts programs needing modification to become elements of multimachine pipeline. Set up prototype parts inventory using INGRES in a separate machine called the Parts Server. [Rice, Gannon, Korb,]	Machine for Parts Server Ethernet interface
i)	Initial work on parts composition system using Megatek graphics system. Solution to the parts connection and type checking problems are due. [Gannon, Snyder, Rice, Tichy]	
j)	Performance evaluation of facility [Denning, Schwetman, Tichy]	
k)	Experiments to evaluate effects of MVP on programming productivity [Dunsmore, Shen]	

PHASE III (1985-87):

a)	Expand MMP to include more machines, integrate with other facilities on campus (e.g., Cray-1). Test under heavy loads. Expand Ethernet bandwidth.	2 VAX for pipeline (Year 4) 2 VAX for pipeline (Year 5)
b)	Install ten workstations per year.	10 workstations (Year 4) 10 workstations (Year 5)
c)	Prototype Quanta parts composition system under construction -- uses MVP stream control functions, Version 2 *tap*, and parts inventory (due by end of Year 4).	
d)	Move Quanta parts composition system to workstations (Year 5).	
e)	Performance evaluation of facility	
f)	Experiments to evaluate Quanta's hypotheses	

5.7 Project Organization and Management

The operational personnel supported by the project include the Director, the Facilities Manager, and two Programmers. The Director will be a member of the faculty supported half time by the project. (Professor Comer has agreed to do this.) The full-time Facilities Manager will report to the Director; he will be responsible for procuring and installing equipment, assisting in the designs, and managing the programmers.

We have requested support for one research associate to assist in the design of the Multiprocess Viewport, the Multimachine Pipeline, and other aspects of the Quanta Environment.

To stimulate better communication with industry, we have requested support for one visitor from industry each year, in exchange for one faculty member who takes sabbatical in industry.

We propose a traineeship program as a direct incentive to attract new PhD students. We have requested support for 5 one-year traineeships for incoming students and 5 one-year renewals. The traineeships are intended to expedite the path through coursework to the PhD qualifying examination. Those who pass the examination will be supported as Research Assistants. Traineeships will be awarded by a screening committee on the basis of promise and merit. To apply for a traineeship, a student must declare his intention to undertake experimental computer science research at the doctoral level. The traineeship program will be nationally advertised.

The proposed facility supports research projects by providing a high-quality environment for those projects. It does not provide direct funding for those projects. Because our ability to keep the facility modern will require us to respond to new opportunities, we have requested a capital fund for equipment we have not foreseen here.

Equipment purchased from this fund would be approved by the project monitor at NSF.

We believe that it is vital to encourage new faculty by reducing teaching loads during the first year and by providing summer support. We have requested funds for four faculty summer grants. Grants will be made by the Department Head on the recommendation of the Project Director.

We believe that the continued success of the project will depend on industrial support. We are therefore exploring with the university methods of accepting industrial grants and sharing the royalties from inventions or sales of software and systems. (All such arrangements would conform to the policies of the NSF.)

6. RESEARCH PROJECTS

While the Multiprocess Viewport and the Multimachine Pipeline are being developed, existing projects will benefit from the prototypes and the increased computer resources. Section 6.1 reviews these benefits. Section 6.2 describes new projects that will compliment or support our approach to distributed computation. Section 6.3 describes projects that will rigorously evaluate the benefits of the Quanta environment. Section 6.4 describes projects that will exploit the Multimachine Pipeline in large scale scientific computation.

6.1 Current Research

6.1.1 The Blue CHiP Architecture Project †

Professor L. Snyder

The goal of the Blue CHiP Project [1] is to exploit the potential of Very Large Scale Integration (VLSI) in the design of a Configurable, Highly Parallel (CHiP) computing system. The main components of the architecture are: a) a large number of simple processing elements (PEs), each with a small but reasonable amount of local memory; b) a front end controller; and c) a lattice of programmable switches in which the PEs are embedded. The switch lattice allows the configuration of data paths between PEs to be established and altered on command from the controller, thereby reducing the amount of data movement during the computation. Because of the regularity of the architecture, VLSI implementation is natural. The configurable switch lattice permits an algorithmic versatility that is not shared by many other contemporary designs [2].

The increased computing power generated by the Multimachine Pipeline project will be essential to three aspects of the initial CHiP development.

1. *Prototype preparation.* We are developing two prototypes of CHiP machines to test different aspects of the architecture. These will be designed using an interactive VLSI design and layout system that will soon be running on the Megatec graphics unit. We will rely heavily on simulation. With the anticipated increase in computing power, we can couple our logic simulator with the CHiP system simulator to validate the system prior to fabrication. (AS the Multimachine Pipeline grows, we expect these components to be distributed over the system.)

---

†ONR Contracts N00014-80-k-0816, "Parallel Computation", and N00014-81-k-0360, "The Blue CHiP Project."

2. *Parallel-language Compilers.* The CHiP approach to parallel computing provides a clear separation between programming the data flow through the switches and programming the PEs. Accordingly we need, and are developing, two nonstandard languages. The first will enable one to program data flow using the Megatec Graphics unit to specify the interconnection structure of the PEs. The second language is for programming PE code; it focuses on ways for the programmer to succinctly specify thousands of nearly identical PE programs. Eventually this work will become part of a CHiP programming environment that makes full use of the Multiprocess Viewport system.

3. *Numerical properties of parallel algorithms.* Numerical problems form a major class of potential appllications for the CHiP machine because they are so compute bound. We are presently studying parallel numerical algorithms for the CHiP architecture. We must run reasonably large problems to evaluate the numerical behavior. To measure the performance of the CHiP system and its software we will build a parallel computation problem solving environment that incorporates the systems described above. This project will collaborate with Gannon's‡ (see below) on parallel numerical methods and with the ELLPACK project of Rice and Lynch.

REFERENCES

[1] Snyder, L., "Introduction the the Configurable, Highly Parallel Computer," CSD-TR-351, Computer Sciences, Purdue University, 1981.

[2] Gannon, D. and Snyder, L., "Linear Recurrence Systems for VLSI: The Configurable, Highly Parallel Approach", *Proceedings International Conference on Parallel Processing*, 1981.

‡NSF Grant MCS-8109512, "Algorithms and Architectures for Highly Parallel Solutions to Partial Differential Equations"

6.1.2 Evaluation of Partial Differential Equations Software †

Professor J. R. Rice

A system [1] for the performance evaluation of partial differential equations software has been constructed based on ELLPACK [2]. The main components of this system are: a) a large population of test problems and an ELLPACK program synthesizer to create specific test sets; b) the ELLPACK system and its 40-odd problem solving modules; c) a data base of performance measures which currently contains data from solving over 6000 elliptic PDEs; and d) a data analysis system to retrieve, analyze, and display information from the data base. This system is in production for performance evaluations [3,4,5]. These evaluations require large amounts of computer time and memory: I hope to start evaluating in the important 3D problems, which will require even more computer resources than the current studies.

REFERENCES

[1] Boisvert, R. F., Rice, J. R. and Houstis, E.N., A system for performance evaluation of partial differential equations software, *IEEE Transactions on Software Eng. 5* (1979), 418-425.

[2] Rice, J. R., ELLPACK: Progress and Plans, in *Elliptic Problem Solvers* (M. Schultz, ed.), Academic Press, (1981), 135-163.

[3] Houstis, E. N. and Rice, J. R., High order methods for elliptic PDEs with singularities, *International J. Numer. Meth. Eng.* (1981).

[4] Rice, J. R., On the effectiveness of iteration for the Galerkin method equations, in *Advances in Computer Methods for PDEs IV* (Vishnevetsky and Stepleman, eds.), IMACS, New Brunswick, N.J. (1981), 60-73.

[5] Rice, J. R., Machine and Compiler effects on the performance of elliptic PDE software, CSD-TR 359, Computer Science, Purdue University, (1981).

†NSF Grant MCS79-01437, "Numerical Solution of Elliptic Patial Differential Equations."

6.1.3 Equations Interpreter †

Professors C. M. Hoffmann and M. J. O'Donnell

Professors Hoffmann and O'Donnell are designing and implementing interpreters for a language whose programs are mathematical equations. Such interpreters are useful for at least four reasons:

- We may write programs as sets of equations. Because the semantics of equations are much simpler than those of normal procedural programming languages, equational programs can be exceptionally easy to understand and verify.
- We may define programming languages by equations; the equation processor will produce interpreters. For instance, the equations from [3], with some corrections, define LISP. An equation interpreter would automatically produce a LISP interpreter which exactly obeys those equations (existing LISP interpreters usually do not).
- Equations describing data types may be used to automatically produce correct implementations as suggested by Guttag, Horowitz, Musser and Wand.
- Theorems of the form A=B may sometimes be proved by receiving the same output when A and B are input, as suggested by Knuth and Bendix.

The theoretical foundations for equation interpreters are developed in Professor O'Donnell's monograph, *Computing in Systems Described by Equations* [4]. Giovanni Sacco and Paul Golick (graduate students) have produced a pilot version of an equation interpreter that employs these ideas and the Hoffman-O'Donnell pattern-matching techniques [1]. The principles of the project are described in *Programming With Equations* [2]. Professor O'Donnell is presently producing a version of the interpreter that can be distributed to other sites. The main needs of this project which could be served

†NSF Grant MCS-78-01812, "A Uniform Theory for Implementations of Descriptive Languages."

by new departmental facilities are:

1. Better support for the user interface through the Quanta Programming Environment; and

2. Better facilities for instrumentation and performance evaluation, through the Multiprocess Viewport.

REFERENCES

[1] Hoffmann, C. M. and O'Donnell, M. J., "Pattern Matching in Trees." To appear in *Journal of the ACM*.

[2] Hoffmann, C. M. and O'Donnell, M. J., "Programming with Equations." To appear in *ACM Trans. on Programming Languages and Systems*.

[3] McCarthy, J., "Recursive Functions of Symbolic Expersions and Their Computation by Machine", Part 1. *Communications ACM 3*, 4 (April 1960), 184-195.

[4] O'Donnell, M. J., *Computing in Systems Described by Equations*, Lecture Notes in Computer Science, #58, Springer-Verlag (1977).

6.1.4 High Order Methods for Elliptic Problems

Professors R.E. Lynch and J.R. Rice

A new high order method [1] for elliptic problems has been found and developed in various ways. It is quite efficient and for certain classes of problems [2]. Theoretical estimates indicate that the method will provide superior performance on complicated elliptic problems that require relatively fine meshes for moderate accuracy. So far we have been unable to verify this theoretical prediction by actual experiments because of the lack of computer resources to systematically evaluate the new method. Such a study is particularly important because this method might have very significant advantages for those PDEs that are currently consuming enormous amounts of computer resources.

REFERENCES

[1] Lynch, R. E. and Rice, J. R., "High accuracy finite difference approximation to solution of elliptic PDEs," *Proc. Nat. Acad. Sci.75* (1978), 2541-2544.

[2] Lynch, R. E. and Rice, J. R., "The Hodie method and its performance," in *Recent Advances in Numerical Analysis* (C. deBoor, ed.) Academic Press (1978), 143-176.

6.2 Systems Design for Distributed Computation

6.2.1 Data Security †

Professor Dorothy E. Denning

In the short term, the expanded facility will benefit our research in methods of protecting confidential data released as summary statistics from disclosure by inference. The INGRES database system would accommodate experimental studies of inference controls; under its present load, our existing facility cannot adequately support this system.

In the long term, we will develop principles and techniques for protecting sensitive data in parts-based distributed systems, using the the Multimachine Pipeline as a testbed. An important component of this research is "security parts" that protect data from unauthorized disclosure and modification. Security parts can be attached to data streams between program parts, between main memory and file storage, or between users (as in mail systems). Security parts will provide a single mechanism for both secure communication and secure storage.

Security parts will use encryption to enforce access control and information flow policies. They will use the DES (or some other high-speed hardware-implemented scheme) in different modes of operation, including stream modes and cipher block chaining. They will use public-key encryption for key distribution and for digital

†NSF Grant MCS-8015484, "Data Security."

signatures. Critical programs will be protected by digitally signing them to prevent substitution of untested parts or Trojan Horses.

The proposed approach departs from traditional approaches to operating systems security by focusing on data streams rather than on objects. It is motivated by the desire for a single mechanism for secure communication and for secure data storage. Most systems use access control mechanisms to protect objects such as segments, files, devices, processes, and extended types. These mechanisms cannot, however, protect transmission lines, which are vulnerable to tapping even when access to the system is denied; encryption is the only means of protecting data transmitted over insecure channels. These mechanisms also cannot protect objects stored on tape or disk which, if stolen, can no longer be protected by the system; here again we need encryption. Encryption also protects against browsing in case of inadequate access controls. By attaching encryption parts to data streams, as in the Multimachine Pipeline, we hope to provide a common collection of parts both for secure communications and for secure storage.

6.2.2 Communications Among Software Parts

Professors W. F. Tichy and T. P. Murtagh

A critical problem in parts-based computing is the communication between software parts. A bewildering variety of communication techniques has developed for programs: data flow, pipes, procedures, coroutines, ports, messages, monitors, communicating sequential processes, rendezvous, and so on. Many of these should be applicable to software parts as well, but without a deeper understanding of them it will be difficult to choose the right ones.

We propose to develop a model of communication for programmed systems, consisting of primitive objects and operations, in order to study communication properties between software parts and to be able to select the best mechanism for each application. We expect the model to show that some of the techniques mentioned above are equivalent. For example, Lauer and Needham [1] have argued informally that message-based systems and a certain style of procedure-based systems are duals.

One important issue in this study will be the kinds of data structures that may be exchanged between software parts. For example, Unix pipes are limited to character strings. We have designed IDL*, an Interface Definition Language, which allows the specification of arbitrary data structures to be exchanged between software parts. (IDL* is a refinement of IDL [2] used for describing intermediate languages for compiler parts.) Besides the primitive data types like rational numbers or strings, IDL* offers records, sets, sequences, and pointers. We will build a special compiler, which, when given an IDL* specification, will generate the prologue and epilogue ("perilogues") for each software part. The perilogues translate to and from a standard data representation (the standard data representation is also determined by the IDL specification) and typecheck the input and output. The perilogues must be tailored automatically to the communication strategy of the parts involved.

Another important issue is the ability to specify software parts that are independent of the details of the communication environment in which they will be used. In the Multimachine Pipeline, many different categories of communication with different levels of overhead will exist. It is tempting to give the programmer different constructs to describe each protocol. This approach has two limitations: it would increase the dependence of programs upon the communication medium; and it would clutter the programming language with many different notations for similar activities. We propose instead that the programmers of parts use a single message-based

communication mechanism in all situations. we will study optimization techniques for each major class of use of a communications mechanism. We will apply our findings to select efficient implementations of the communciation primitives used by the parts.

REFERENCES

[1] Goos G., and Wulf, W. A., *Diana Reference Manual*, Technical Report, Carnegie-Mellon University, Computer Science Department March 1981.

[2] Lauer, H. C., and Needham, R. M., "On the Duality of Operating System Structures," *Proc. IRIA Conference on Operating Systems*, North-Holland Publishing Co., October 1978.

6.2.3 The Synchronization of Parallel Computation

Professors J. Cuny and L. Snyder

The Multimachine Pipeline facility and the Blue CHiP processors are representatives of the two fundamental approaches to parallelism: the MMP is asynchronous while the Blue CHiP is synchronous. The asynchronous approach is more natural for data driven, general purpose computation and may be more efficient for some computational tasks [2]; but in many cases there are advantages to the synchronous approach [1,3]. We intend to formalize the relation between these approaches and to investigate their capabilities and limitations. Our results will provide aid the design of the large collections of software that have been proposed for the MMP and Blue CHiP systems. In addition, we expect to develop algorithms that map programs between systems based on the different forms of parallelism.

REFERENCES

[1] Arjomandi, E., Fischer, M. J., and Lynch, N, "A difference in efficiency between synchronous and asynchronous systems," Technical Report #81-03-01, Dept. of Computer Science, University of Washington, 1981.

[2] Gannon, D. B., "On Mapping Non-uniform P.D.E. Structures onto Uniform Array Architectures," *Proc. 1981 International Conference on Parallel Processing*. IEEE Cat. no. 81CH1634-5.

[3] Lipton, R. J., Miller, R. E., and Snyder, L., "Synchronization and computing capabilities of linear asynchronous structures," *JCSS 14*, 1 (February 1977), 49-72.

6.2.4 Performance Bounds of Multiprocessor Systems

Professor H. D. Schwetman

Schwetman and Reed (graduate student) have investigated asymptotic bounds on the throughput rates of multiprocessor systems operating under heavy load. The type of system is a collection of many (hundreds or thousands) processing elements, each with its own local memory and one or more communications ports. These processing elements are interconnected via communication links. So far, the modeling effort has evaluated eleven different interconnection schemes, producing estimates of transaction processing rates and relative costs of each scheme, when the system is saturated. The parameters of the model include the number of processing elements, the mean processor and link service times, as well as the interconnection scheme. The initial work is reported in a technical report, which is being submitted for consideration in a special issue of the IEEE *Transactions on Computer Systems*.

We propose to explore the following topics, in the performance of a network of aysnchronous distributed processing elements:

1. What are optimal distributions of work over a network of processing elements?
2. When parallel computational tasks are being designed, what is a "good" level of granularity?
3. What techniques can be used to ascertain the status of elements near another element and far away from another element?
4. What factors affect the decision to forward work on to an adjacent element, as opposed to processing the work locally?

Simulation is an attractive approach for the initial research. Later, when some principles emerge, we hope to be able to use queueing network models and asymptotic bounds to obtain functional relationships.

6.2.5 Programming Support on Personal Computer Networks

Professor W. F. Tichy

A programming support environment (PSE) aids in the development and maintenance of programmed systems. Currently, most such environments run on single, timeshared computers with central storage facilities. On a personal computer network, new problems arise because the project information is no longer centralized; it is spread over the local storage in the personal computers and over one or several file servers.

We propose to study how components of programming support environments can be distributed over personal computer networks. We will build a prototype PSE to evaluate our ideas.

There are three major problem areas: *data base design*, *version control*, *and interface control*. The data base is a critical component of a PSE, because it is the major data structure for depositing all project information and is accessed by all tools. Previous research has shown that a directed, attributed graph is an adequate, high-level view of a centralized PSE data base. We plan to extend this notion to distributed PSEs.

Version control is concerned with keeping a system family comprising many versions and configurations in a well-defined state. Interface control establishes and maintains consistent interfaces between the many interfaces. We plan to design the mechanisms and algorithms for performing these two functions in a distributed environment. Because of the decentralization, we must develop incremental mechanisms for these.

6.2.6 Interactive Programming Environments

Professor J. T. Korb

This research centers on the design and implementation of an interactive programming environment. The emphasis is on a practical, efficient system that can be used to develop both small and large programs. This research explores such issues as non-linear languages (i.e., freedom from stream-oriented parsers), incremental compiling and loading, reversible execution, and source-level dynamic program modification and data structure manipulation.

An ordinary CRT (or "dumb terminal") is a poor I/O device for programming environment research for at least three reasons: 1) the screen size is too small and not dense enough to allow presentation of very much information to the user; 2) there is no simple method (other than cursor control keys) to reference or select items presented on the screen; and 3) the relatively narrow bandwidth between CRT and host computer, as well as the varying load on the host computer, prevents rapid changes to the display. The large-screen display, multiple windows, and process control facilities of the multiprocess viewport will provide the necessary tools for this research.

While at Xerox PARC, Professor Korb developed techniques for the manipulation of text on a high-resolution bitmap display and, later, led the implementation effort of a production-quality programming environment. He intends to apply this experience to

the development of new and better program development tools.

6.3 The Quanta Programming Environment: Design and Analysis

6.3.1 Verifying the Utility of Parts-Based Programming †

Professor H. Dunsmore

It is important to gather empirical evidence concerning the Quanta programming environment. Our hypothesis is that programmers will be more productive in such an environment than in an ordinary one. We will subject this hypothesis to experimentation by comparing the performance of subjects constructing equivalent software in the two environments. Critical measures for statistical analysis will be time, number of errors committed, quality of the finished product, etc. Post-experiment debriefings will provide further insight into the reasons for the differences we find. We have experience in designing, conducting, and analyzing experiments of the type that we will employ [1,2,3].

A similar technique will be employed to make the best choices for features embedded in the Quanta environment. For example, Quanta graphs will be used to specify parts networks. The experimental nature of this research allows us to investigate whether different forms of Quanta graphs affect productivity. Subjects would be asked to construct equivalent software with the variants. This will allow us to select implementation techniques that can be shown empirically to reduce time and errors.

Another area in which experimentation will play a major role in our research is in the interface between user and system. The Quanta environment allows one to apply

---

†Grants by IBM Corporation: "Testing of an Interactive Facility for Non-Trained Users," and "Testing of an Interactive System to Include Package Execution Capability."

existing software parts to data. This requires a very high level language to help select appropriate parts and specify the details of executing them. We anticipate that parts-based programming will appeal to users along a continuum from experienced programmers to inexperienced nonprogrammers. We will conduct experiments designed to refine the interaction medium for maximum productivity. We have been conducting research on this topic and have determined parameters that will be useful in defining a language for non-programmers [4]. The techniques developed in our previous research can be adapted for interaction media for experienced programmers.

REFERENCES

[1] Dunsmore, H., "Experience with empirical research in human factors," *Proceedings of SHARE 57*, Chicago, Illinois (August 1981).

[2] Woodfield, S., Dunsmore, H. and Shen, V., "The effect of modularization and comments on program comprehension," *Proceedings of Fifth International Conference on Software Engineering*, San Diego, California (March 1981), 215-223.

[3] Dunsmore, H. and Gannon, J., "Analysis of the effects of programming factors on programming effort," *The Journal of Systems and Software 1*, 2 (February, 1980), 141-153. Also in *Models and Metrics for Software Management and Engineering*, (V. R. Basili, ed.), Computer Society Press, Los Alamitos, Calif. (1980), 93-105.

[4] Dunsmore, H., "Designing an interactive facility for non-programmers," *Proceedings of ACM 80*, Nashville, Tennessee (October 1980), 475-483.

6.3.2 Programming Effort for Very High Level Languages

Professors J. R. Rice and V. Shen

A basic hypothesis of the Quanta environment is that Very High Level (VHL) languages and Problem Solving Environments (PSEs) will greatly reduce the programming effort required for a particular task. This hypothesis cannot be verified with certainty until a prototype Quanta system is operational. The study by Dunsmore will provide concrete, measured data to test this hypothesis. To complement Dunsmore's effort, software science methodology [1] will be extended to very high level languages. We plan to refine the notion of "power" of a programming language [2] using software

science parameters as well as simple paramenters such as lines of code. This approach will provide a tool to estimate the programming effort for VHL languages which can be applied to the Quanta environment. Pilot studies [3,4] indicate that the software science methodology does extend to VHL languages nicely. We expect to obtain good correlations with the data of Dunsmore's effort and thus provide predictors of programming effort that are as reliable as those software science now provides for convential Fortran/Cobol programming [1].

REFERENCES

[1] Shen, V. Y., Conte, S. D., and Dunsmore, H. E., "Software Science Revisited", report in preparation, Computer Science, Purdue Univ., (1981).

[2] Rice, J. R., "Programming Languages: Power, Trends and Facilities for Numerical Computation," in *Relationship Between Programming Languages and Numerical Computation* (J. Reid, ed.), North-Holland (1982).

[3] Rice, J. R., "Programming effort analysis of the ELLPACK language," ACM *SIGNUM Newsletter 14* (1979), 109-111.

[4] Rice, J. R. and Shen, V. Y., "Software Science Analysis of a Problem Solving Extension of Fortran," report in preparation, Computer Science, Purdue University, (1981).

6.3.3 A Problem Solving Environment for Queueing Networks

Professor H. D. Schwetman

Queueing network models are useful as models of computer systems. They apply today to a wide variety of common systems. There is now a need to bring the fruits of this research to the desks of systems analysts.

We propose to develop a problem solving environment for systems analysts. This will aid in the design and analysis of computer and communications systems. A user could begin a system design by creating a diagram of the proposed system on a graphics device. He would associate parameters with the paths and nodes of the system. He would obtain performance estimates for the design by letting the PSE solve the underlying queueing network model. The design could be modified and solved repeatedly

until satisfactory. The system would include special facilities for large networks, a file system for storing and retreiving different models and versions of models, and facilities for comparing the performance of different versions. Output would consist of tables and plots of selected performance variables as functions of the load and of system features.

Measurements of a user at work in this PSE will quantify the benefits to be obtained from it.

6.3.4 An Environment for Solving Partial Differential Equations

Professor J. R. Rice

The ELLPACK system [1] is a very high level language for stating elliptic partial differential equations (PDEs) and specifying how they are to be solved (e.g. one says "use finite differences and Gauss elimination" or "use Galerkin with Hermite cubics and SOR"). This system is now being implemented by a Fortran preprocessor as an extension of Fortran. Because it permits one to move freely between the elliptic PDE environment and the Fortran environment, this is a nascent Problem Solving Environment. From these modest beginnings we plan a) to make the PDE environment and Fortran environment more independent without sacrificing the ability to move freely, b) to extend the scope of the system to include a third environment, graphical output; and c) to explore various other extensions (e.g. time dependent PDEs or non linear problems) and to study their impact on the nature of the problem solving environment (one wants to keep the language natural and simple).

REFERENCE

[1] Rice, J. R., "ELLPACK: Progress and Plans," in *Elliptic Problem Solvers* (M. Schultz, ed.), Academic Press, (1981), 135-163.

6.3.5 Rapid Implementation of PSE Language Processors †

Professor J.R. Rice

The Quanta Project envisages a computing environment with many problem solving environments available. These environments normally combine a simple, natural language with a collection of software parts to solve the problem specified. We are developing a technique for implementing the language processors in a systematic and automatic way. While this approach is resembles the compiler-compiler technique, we differ in the following respects: a) a style of language description (grammar) is used which is much closer to the language being defined and hence much easier to specify; b) the translation is to an algorithmic language such as Fortran; c) the grammar uses fact b) so that, for example, an "expression" is a known item; d) programs are assumed to be short and have simple structure so that efficiency in the language processing is not very important and complex control structures need not be supported. A prototype of this methodology is now being tested [1] this project is part of the TOOLPACK [2] effort to support Fortran programming.

REFERENCES

[1] Brophy, J., "Problem Oriented Language Preprocessor Generator," report in preparation, Computer Science, Purdue University, 1981.

[2] Cowell, W. R. and Miller, W. C., "The TOOLPACK Prospectus," Appl. Math TM-341, Argonne Nat. Lab. (1981).

---

†NSF Grant MCS79-01437, "Template Processors and Toolpack Evaluation."

6.4 Scientific Computation on the Multimachine Pipeline

6.4.1 Symbolic Mathematical Computation

Professor D. Arnon

The Multimachine Pipeline will be used for studies of parallelism in symbolic computation and combined symbolic-numeric computation. Our objectives are, first, to make certain fundamental symbolic algorithms, such as symbolic integration, polynomial factorization, and algebraic number arithmetic, available as software parts, and, second, to explore schemes of composing these parts to obtain efficient, parallel versions of more complex algorithms. We also hope to draw on available numerical software to investigate combinations of symbolic and numerical algorithms. The Multimachine Pipeline and the Quanta Programming Environment will provide a framework for these investigations.

6.4.2 Networked Numerical Algorithms †

Professor D. Gannon

The MMP and the Quanta programming environment provide both a formal foundation and a structural framework for synthesizing distributed numerical algorithms. Rather than developing algorithms by extracting low level parallelism from existing software, it is possible to compose higher level sequential parts into networks of interacting processes. For example, the solution of a large block tridiagonal system of equations may be accomplished by the interconnection of a distributed collection of well understood band system solvers This research will be organized into two phases of

---

†NSF Grant MCS-8109512, "Algorithms and Architectures for Highly Parallel Solutions to Partial Differential Equations."

experiments. First we will test the viability of the concept by designing a networked version of a large hydrodynamic code (such as SIMPLE [1]). In this experiment we will be concerned with both the utility of the Quanta programming tools as well as the multiprocessor performance.

In the second phase we will concentrate on adaptive computation. As observed by Zave and Rheinboldt [2], it is reasonalbe to design an adaptive elliptic PDE solver around distributed computation and user interaction. In this scheme, the local behavior of refined grid structures is determined by certain numerical indicators produced as by-products of the computation. We propose to build such a system using the the Multiprocess Viewport to monitor the local indicators so that the user will then have the ability to guide the adaptive process. A closely related project is described by Professor McCarthy below.

REFERENCES

[1] "SIMPLE": A benchmark for parallel computation. Lawrence Livermore National Laboratory.

[2] Zave, P. and Rheinboldt, W., "Design of an Adaptive, Parallel Finite-Element System", *ACM Trans. on Math. Software 5*, 1 (1979), 1-17.

6.4.3 Parallelism in Computational Fluid Dynamics

Professor D. McCarthy

Modest first steps in the exploitation of parallel (and pipeline) processing in Computational Fluid Dynamics (CFD) have already been made possible by the appearance of vector machines (Cray-1, ILIAC IV, CYBER 203). So far, research revolves around algorithm design with the objective of achieving effective vectorization of existing schemes. Any system which enhances the degree and flexibility of parallel computa-

tion must have profound effect on algorithm development.

At a higher level, however, actual multi-machine interactiveprocessing could enable radically different approaches to currently intractable problems. To cite two:

1. The relatively new and powerful "multigrid" method can be adapted to spawn decoupled problems on highly refined spatially disjoint grids, which could then be processed simultaneously. Recombination via the MVP could then lead to highly accurate global solutions.
2. The notion of decoupling is often extended to spatially separate flow regions in which different equation sets (boundary layer, inviscid potential, etc.) are solved. Recombination is currently achieved iteratively, limiting most practical applications to two regions. Interactive comparison of simultaneously computed results for different flow regions could lead to much more effective procedures, and bring currently intractable problems within reach.

7. APPENDIX A – SPONSORED RESEARCH PROJECTS

F. Berman, 6/1/80 - 5/31/82, "Theoretical and Practical Considerations in Propositional Dynamic Logic," NSF, $25,659.

D. Comer, 6/1/81 - 5/31/82, "Language and Tools for String Processing," Donnelly & Sons, $21,911.

S. Conte, 4/6/76 - indef., "Research in Software Physics," GM Laboratories, $10,045.

S. Conte & V. Shen, 1/1/80 -12/31/82, "Research on Software Science," IBM, $89,107.

D. Denning, 1/1/81 - 12/31/82, "Data Security," NSF, $63,150.

P. Denning & J. Buzen, 8/1/78 - 12/31/81, "Operational Analysis of Queueing Phenomena," NSF Industry/University Cooperative, $230,403.

H. Dunsmore, 9/9/79 - 9/8/81, "Experimental Investigation of Programming Complexity in COBOL," ARO, $62,214.

H. Dunsmore, 8/1/79 - indef., "Testing of an Interactive Facility for Non-Trained Users," IBM, $27,737.

H. Dunsmore, 1/1/81 - indef., "Extending an Interactive System to Include Package Execution Capability," IBM, $56,686.

D. Gannon, 6/1/81 - 5/31/84, "Algorithms and Architectures for Highly Parallel Solutions to Partial Differential Equations," NSF, $178,665.

W. Gautschi, 6/1/80 - 5/31/82, "Gauss Type Quadrature Rules," NSF, $41,275.

C. Hoffman & M. O'Donnell, 9/1/78 - 2/28/83, "A Uniform Theory for Implementations of Descriptive Languages," NSF, $243,974.

J. Rice & R. Lynch, 9/1/76 - 8/31/81, "Numerical Solution of Elliptic Partial Differential Equations," NSF, $148,364.

J. Rice & R. Lynch, 6/1/79 - 8/31/82, "Computer Network for Numerical Analysis Research," NSF, $24,219.

J. Rice, 7/1/80 - 6/30/82, "Template Processors and Toolpack Evaluation," NSF, $87,011.

J. Rice, 7/1/80 - 6/30/82, "Nonlinear Approximation," NSF, $43,372.

C. Smith & P. Young, 8/1/80 - 11/30/81, "Workshop on Recursion Theoretic Aspects of Computer Science," NSF, $9,400.

L. Snyder, 6/1/80 - 5/31/83, "Parallel Computation," ONR, $122,500.

L. Snyder, 1/1/81 -12/31/83, "The Blue CHiP Project," ONR, $825,000.

L. Synder, 2/1/81 - 1/31/82, "Secure Computation," NSF, $38,418.

W. Tichy, 6/1/81 - 5/31/83, "Research on Configuration Management for Programmed Systems," NSF, $40,754.

P. Young, 6/1/76 - 9/30/81, "Mathematical Theory of Computation," NSF, $224,446.

Total of Above Projects: $2,620,324.